Python Code for HFI Model

Listing 1: Python Code for HFI Model

```
import numpy as np
import cmath
import csv
from scipy.constants import hbar

class HFIModel:
    def __init__(self):
        # Fundamental constants
        self.MH = 125.1  # Higgs mass in GeV
        self.lambda_pc = 0.00464  # Pythagorean comma coupling
        self.pythagorean_comma = 1.013643
        self.tau0 = hbar / (1e9 * 1.60218e-19)  # Natural timescale
             in GeV^-1

    def compute_properties(self, M):
        """Compute all particle properties from mass M"""
        h = np.log2(self.MH / M)

        # Charge calculation
        angle = 2 * np.pi * h
        pc_term = self.lambda_pc * (self.pythagorean_comma**(h/12)
            - 1)
        with np.errstate(all='ignore'):
            Q = np.sin(angle) - np.cos(angle) - np.tan(angle) +
                pc_term
            Q = np.real(Q) if not np.isnan(Q) else 0

        # Spin and helicity
        S = 0.5 * (1 + np.cos(2 * np.pi * h))
        helicity = S * np.sign(np.sin(2 * np.pi * h))

        # Angular momentum components
        Jx = S * np.cos(2 * np.pi * h / 3)
        Jy = S * np.sin(2 * np.pi * h / 3)
        Jz = S * np.cos(np.pi * h)

        # Lifetime calculation
        with np.errstate(all='ignore'):
            lifetime = self.tau0 / (np.sin(angle) - np.tan(angle))
            lifetime = lifetime if not (np.isinf(lifetime) or np.
                isnan(lifetime)) else None

        # Charge quantization
        n = round(h * 12)
        Q_quantized = self.quantized_charge(n)
        charge_group = self.charge_group(n)
        spin_statistics = 'fermion' if n % 2 else 'boson'

        return {
            'particle': None,  # To be filled later
            'mass_GeV': M,
            'h': h,
            'Q_operator': Q,
```

```
            'Q_quantized': Q_quantized,
            'S_operator': S,
            'helicity': helicity,
            'Jx': Jx,
            'Jy': Jy,
            'Jz': Jz,
            'lifetime': lifetime,
            'charge_group': charge_group,
            'spin_statistics': spin_statistics
        }

    def quantized_charge(self, n):
        """Exact charge quantization for h = n/12"""
        n_mod = n % 12
        if n_mod in {0, 4, 8}:
            return 2/3  # Up-type quarks
        elif n_mod in {3, 7, 11}:
            return -1/3  # Down-type quarks
        elif n_mod in {1, 5, 9}:
            return -1  # Leptons
        else:
            return 0  # Neutral particles

    def charge_group(self, n):
        """Particle classification by charge"""
        n_mod = n % 12
        if n_mod in {0, 4, 8}:
            return 'up-type quark'
        elif n_mod in {3, 7, 11}:
            return 'down-type quark'
        elif n_mod in {1, 5, 9}:
            return 'lepton'
        else:
            return 'boson'

# Particle data
particles = {
    # Leptons
    "electron": 0.000511,
    "muon": 0.10566,
    "tau": 1.77686,

    # Quarks
    "up_quark": 0.0022,
    "down_quark": 0.0047,
    "strange_quark": 0.096,
    "charm_quark": 1.27,
    "bottom_quark": 4.18,
    "top_quark": 173.0,

    # Gauge Bosons
    "W_boson": 80.379,
    "Z_boson": 91.1876,
    "Higgs_boson": 125.1,
}

# Initialize and compute
```

```
hfi = HFIModel()
results = []

for name, mass in particles.items():
    props = hfi.compute_properties(mass)
    props['particle'] = name
    results.append(props)

    # Print summary
    print(f"\n--- {name} (M = {mass} GeV) ---")
    print(f"h = {props['h']:.5f}")
    print(f"Group: {props['charge_group']}")
    print(f"Q (quantized) = {props['Q_quantized']}")
    print(f"Spin = {props['S_operator']:.3f} ({props['
        spin_statistics']})")
    print(f"Lifetime = {props['lifetime'] if props['lifetime'] is
        not None else 'stable'} GeV^-1")

# Define CSV columns matching our computed properties
csv_columns = [
    'particle', 'mass_GeV', 'h',
    'Q_operator', 'Q_quantized', 'S_operator',
    'helicity', 'Jx', 'Jy', 'Jz',
    'lifetime', 'charge_group', 'spin_statistics'
]

# Export to CSV
csv_file = "hfi_particle_properties.csv"
with open(csv_file, 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=csv_columns)
    writer.writeheader()
    for data in results:
        writer.writerow(data)

print(f"\nResults saved to {csv_file}")
```